Универзитет у Београду

Електротехнички факултет



Двопролазни асемблер

Системски софтвер

Страхиња Стефановић

2016/0130

Београд, јун 2019.

# Опис проблема

Задатак је направити двопролазни асемблер вођен принципима GNU асемблера. Асемблер у првом пролазу генерише табелу симбола, а у другом генерише кодове инструкција и формира релокационе записе за оне симболе чија се вредност не зна на крају процеса асемблирања.

Излаз асемблера чине два фајла. Први је бинарни објектни фајл са екстензијом .o који ће се касније користити као улазни параметар емулатора, а други је стандардни текстуални фајл чија је улога да прикаже принцип рада асемблера и његову коректност.

# Упутство за превођење и покретање

Да би програм могао да се преведе и извршава потребно је најпре инсталирати новију верзију g++ компајлера. То је најлакше урадити покретањем приложене скрипте commands.sh.

За превођење програма користи се makefile скрипта:

make ./bin/assembler

која генерише извршни фајл у одговарајућем директоријуму.

Сам програм потребно је покренути на следећи начин:

./bin/assembler –o <output\_file> <input\_file>

Улазни фајл представља асемблерски програмски кôд, а излазни фајл назив жељеног предметног фајла. Уколико се не проследи одговарајући број аргумената и то на овај начин, асемблер ће пријавити грешку. Такође, биће грешка и ако улазни и излазни фајлови немају одговарајуће екстензије, .s, односно .o.

За извршавање датих тестова потребно је покренути скрипту test.sh.

# Опис решења

Улазни фајл се најпре издели на токене, притом игноришући коментаре и све након директиве .end.

Након тога се извршава први пролаз у којем се сви симболи на које се наиђе стављају у табелу симбола и, евентуално, ако је реч о секцији, у табелу секција. Осим тога, за сваку инструкцију у кôду се формира објекат класе Instruction која садржи све потребне податке за лакше генерисанје машиског кôда у другом пролазу. У овом тренутку ће асемблер одмах пријавити грешку уколико је формат инструкције невалидан.

У другом пролазу се формира машински кôд свих секција (осим оних које нису назначене као allocatable) уз генерисање записа о релокацијама на оним местима где није позната тачна вредност симбола. За локалне симболе дефинише се релокација у односу на почетак секције у којој се налази, а за глобалне у односу на сам тај симбол. Разлог томе је тај што линкер не зна за локалне симболе унутар фајла (само теоријски пошто се у овој реализацији асемблера у табелама симбола сваког фајла и даље налазе локални симболи).

У реализацији коришћене су помоћне класе:

* Symbol – улаз у табелу симбола
* UnresolvedSymbol – структура која памти зависности симбола дефинисаног .equ директивом од симбола којим је дефинисан
* Section – улаз у табелу секција
* Relocation – релокациони запис
* Instruction – структура која чува операнде и све податке везане за инструкцију (величина, тип инструкције и сл.)

# Тестови

1. Исправност директива .align, .skip, .byte и .word

Улаз : directives.s



Излаз : directives.txt



1. Исправност .equ директивe

Улаз : equ.s



Излаз : equ.txt



1. Детекција цикличне еквиваленције

Улаз : equ\_cycle.s



Излаз :

ASSEMBLING ERROR(line 8): Cyclic equivalence detected!

1. Начини адресирања

Улаз : addressing.s



Излаз : addressing.txt



1. Директива .global

Улаз : global.s



Излаз : global.txt



1. Инструкције скока

Улаз : jumps.s



Излаз : jumps.txt



1. Прекидне рутине

Улаз : setup.s

.section iv\_table, "a"

.word ivt\_entry0

.word ivt\_entry1

.word ivt\_entry2

.word ivt\_entry3

.skip 8

.section routines, "ax"

.global ivt\_entry0, ivt\_entry1

.global ivt\_entry2, ivt\_entry3

.extern \_start

# processor initialization

ivt\_entry0:

mov r0, 0

mov r1, 0

mov r2, 0

mov r3, 0

mov r4, 0

mov r5, 0

mov sp, 0xFF00 # stack start

mov \*0xFF10, 0 # reset timer\_cfg

mov psw, 0xE000 # allow all interrupts

mov pc, &\_start # jump to address of the first instruction

# invalid instruction

ivt\_entry1:

mov \*0xFF00, 73 # I

mov \*0xFF00, 110 # n

mov \*0xFF00, 97 # a

mov \*0xFF00, 108 # l

mov \*0xFF00, 105 # i

mov \*0xFF00, 118 # v

mov \*0xFF00, 100 # d

mov \*0xFF00, 32 #

mov \*0xFF00, 105 # i

mov \*0xFF00, 110 # n

mov \*0xFF00, 115 # s

mov \*0xFF00, 116 # t

mov \*0xFF00, 114 # r

mov \*0xFF00, 117 # u

mov \*0xFF00, 99 # c

mov \*0xFF00, 116 # t

mov \*0xFF00, 105 # i

mov \*0xFF00, 111 # o

mov \*0xFF00, 110 # n

mov \*0xFF00, 33 # !

mov \*0xFF00, 10 # \n

halt

# timer interrupt routine

ivt\_entry2:

# mov \*0xFF00, 84 # T

# mov \*0xFF00, 105 # i

# mov \*0xFF00, 109 # m

# mov \*0xFF00, 101 # e

# mov \*0xFF00, 114 # r

# mov \*0xFF00, 32 #

# mov \*0xFF00, 116 # t

# mov \*0xFF00, 105 # i

# mov \*0xFF00, 99 # c

# mov \*0xFF00, 107 # k

# mov \*0xFF00, 101 # e

# mov \*0xFF00, 100 # d

# mov \*0xFF00, 33 # !

# mov \*0xFF00, 10 # \n

# mov \*0xFF10, 3

iret

# terminal interrupt routine

ivt\_entry3:

mov \*0xFF00, \*0xFF02

mov \*0xFF02, 0

iret

Излаз : setup.txt

